1. **Arrays -**
   1. Array is defined as an ordered set of similar data items. All the data items of an array are stored in consecutive memory locations in RAM. The elements of an array are of the same data type and each item can be accessed using the same name.

Reference - [https://www.scaler.com/topics/data-structures/array-data-structure/](https://airlock-on-edge.woolf.university/?url=https%3A%2F%2Fwww.scaler.com%2Ftopics%2Fdata-structures%2Farray-data-structure%2F&resourceId=98568230-8e4d-43be-ba50-6df27593b8b7&studentId=54badd31-0e83-4844-9fd0-1b7a46e1dea3&token=eyJhbGciOiJIUzI1NiJ9.eyJpZCI6IjU0YmFkZDMxLTBlODMtNDg0NC05ZmQwLTFiN2E0NmUxZGVhMyIsImlzcyI6InVybjpXb29sZlVuaXZlcnNpdHk6c2VydmVyL3NlcnZpY2UvYWNjZXNzIiwiaXNWZXJpZmllZCI6dHJ1ZSwia2luZCI6Im9hdXRoIiwib3JnIjp7Imdyb3VwcyI6W10sImlkIjoiOWIxN2Y1Y2UtMTA3OC00ZmRmLWFlYzAtMDJiZjRlY2ZiMGE2In0sInNjb3BlIjoiKiJ9.hABJPfI6sMVKc09uSyq_3iyl1uWJtLTFmEMua41yAl0)

1. **Why do arrays have O(1) access time?**

An array starts at a specific memory address. Each element occupies the same amount of bytes element\_size. The array elements are located one after another in the memory from the start address on. So you can calculate the memory address of the element i with start + i \* element\_size. This computation is independent of the array size and is therefore O(1).

Ref - [https://stackoverflow.com/a/23103837](https://airlock-on-edge.woolf.university/?url=https%3A%2F%2Fstackoverflow.com%2Fa%2F23103837&resourceId=98568230-8e4d-43be-ba50-6df27593b8b7&studentId=54badd31-0e83-4844-9fd0-1b7a46e1dea3&token=eyJhbGciOiJIUzI1NiJ9.eyJpZCI6IjU0YmFkZDMxLTBlODMtNDg0NC05ZmQwLTFiN2E0NmUxZGVhMyIsImlzcyI6InVybjpXb29sZlVuaXZlcnNpdHk6c2VydmVyL3NlcnZpY2UvYWNjZXNzIiwiaXNWZXJpZmllZCI6dHJ1ZSwia2luZCI6Im9hdXRoIiwib3JnIjp7Imdyb3VwcyI6W10sImlkIjoiOWIxN2Y1Y2UtMTA3OC00ZmRmLWFlYzAtMDJiZjRlY2ZiMGE2In0sInNjb3BlIjoiKiJ9.hABJPfI6sMVKc09uSyq_3iyl1uWJtLTFmEMua41yAl0)

1. **Dynamic Arrays in different languages -**

c++ (Vectors): [https://www.scaler.com/topics/cpp/vector-in-cpp/](https://airlock-on-edge.woolf.university/?url=https%3A%2F%2Fwww.scaler.com%2Ftopics%2Fcpp%2Fvector-in-cpp%2F&resourceId=98568230-8e4d-43be-ba50-6df27593b8b7&studentId=54badd31-0e83-4844-9fd0-1b7a46e1dea3&token=eyJhbGciOiJIUzI1NiJ9.eyJpZCI6IjU0YmFkZDMxLTBlODMtNDg0NC05ZmQwLTFiN2E0NmUxZGVhMyIsImlzcyI6InVybjpXb29sZlVuaXZlcnNpdHk6c2VydmVyL3NlcnZpY2UvYWNjZXNzIiwiaXNWZXJpZmllZCI6dHJ1ZSwia2luZCI6Im9hdXRoIiwib3JnIjp7Imdyb3VwcyI6W10sImlkIjoiOWIxN2Y1Y2UtMTA3OC00ZmRmLWFlYzAtMDJiZjRlY2ZiMGE2In0sInNjb3BlIjoiKiJ9.hABJPfI6sMVKc09uSyq_3iyl1uWJtLTFmEMua41yAl0)

Java (ArrayList): [https://www.interviewbit.com/problems/arraylist/](https://airlock-on-edge.woolf.university/?url=https%3A%2F%2Fwww.interviewbit.com%2Fproblems%2Farraylist%2F&resourceId=98568230-8e4d-43be-ba50-6df27593b8b7&studentId=54badd31-0e83-4844-9fd0-1b7a46e1dea3&token=eyJhbGciOiJIUzI1NiJ9.eyJpZCI6IjU0YmFkZDMxLTBlODMtNDg0NC05ZmQwLTFiN2E0NmUxZGVhMyIsImlzcyI6InVybjpXb29sZlVuaXZlcnNpdHk6c2VydmVyL3NlcnZpY2UvYWNjZXNzIiwiaXNWZXJpZmllZCI6dHJ1ZSwia2luZCI6Im9hdXRoIiwib3JnIjp7Imdyb3VwcyI6W10sImlkIjoiOWIxN2Y1Y2UtMTA3OC00ZmRmLWFlYzAtMDJiZjRlY2ZiMGE2In0sInNjb3BlIjoiKiJ9.hABJPfI6sMVKc09uSyq_3iyl1uWJtLTFmEMua41yAl0)

Concept of Autoboxing and Unboxing in Java - [https://slides.com/tarunluthra/autoboxing-java](https://airlock-on-edge.woolf.university/?url=https%3A%2F%2Fslides.com%2Ftarunluthra%2Fautoboxing-java&resourceId=98568230-8e4d-43be-ba50-6df27593b8b7&studentId=54badd31-0e83-4844-9fd0-1b7a46e1dea3&token=eyJhbGciOiJIUzI1NiJ9.eyJpZCI6IjU0YmFkZDMxLTBlODMtNDg0NC05ZmQwLTFiN2E0NmUxZGVhMyIsImlzcyI6InVybjpXb29sZlVuaXZlcnNpdHk6c2VydmVyL3NlcnZpY2UvYWNjZXNzIiwiaXNWZXJpZmllZCI6dHJ1ZSwia2luZCI6Im9hdXRoIiwib3JnIjp7Imdyb3VwcyI6W10sImlkIjoiOWIxN2Y1Y2UtMTA3OC00ZmRmLWFlYzAtMDJiZjRlY2ZiMGE2In0sInNjb3BlIjoiKiJ9.hABJPfI6sMVKc09uSyq_3iyl1uWJtLTFmEMua41yAl0)

Python (List): [https://www.scaler.com/topics/python/list-in-python/](https://airlock-on-edge.woolf.university/?url=https%3A%2F%2Fwww.scaler.com%2Ftopics%2Fpython%2Flist-in-python%2F&resourceId=98568230-8e4d-43be-ba50-6df27593b8b7&studentId=54badd31-0e83-4844-9fd0-1b7a46e1dea3&token=eyJhbGciOiJIUzI1NiJ9.eyJpZCI6IjU0YmFkZDMxLTBlODMtNDg0NC05ZmQwLTFiN2E0NmUxZGVhMyIsImlzcyI6InVybjpXb29sZlVuaXZlcnNpdHk6c2VydmVyL3NlcnZpY2UvYWNjZXNzIiwiaXNWZXJpZmllZCI6dHJ1ZSwia2luZCI6Im9hdXRoIiwib3JnIjp7Imdyb3VwcyI6W10sImlkIjoiOWIxN2Y1Y2UtMTA3OC00ZmRmLWFlYzAtMDJiZjRlY2ZiMGE2In0sInNjb3BlIjoiKiJ9.hABJPfI6sMVKc09uSyq_3iyl1uWJtLTFmEMua41yAl0)

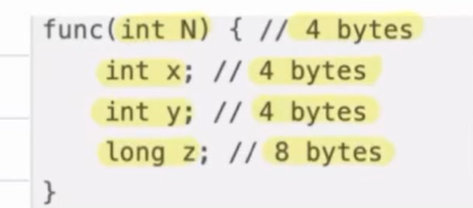
Javascript (Array): [https://www.scaler.com/topics/array-methods-in-javascript/](https://airlock-on-edge.woolf.university/?url=https%3A%2F%2Fwww.scaler.com%2Ftopics%2Farray-methods-in-javascript%2F&resourceId=98568230-8e4d-43be-ba50-6df27593b8b7&studentId=54badd31-0e83-4844-9fd0-1b7a46e1dea3&token=eyJhbGciOiJIUzI1NiJ9.eyJpZCI6IjU0YmFkZDMxLTBlODMtNDg0NC05ZmQwLTFiN2E0NmUxZGVhMyIsImlzcyI6InVybjpXb29sZlVuaXZlcnNpdHk6c2VydmVyL3NlcnZpY2UvYWNjZXNzIiwiaXNWZXJpZmllZCI6dHJ1ZSwia2luZCI6Im9hdXRoIiwib3JnIjp7Imdyb3VwcyI6W10sImlkIjoiOWIxN2Y1Y2UtMTA3OC00ZmRmLWFlYzAtMDJiZjRlY2ZiMGE2In0sInNjb3BlIjoiKiJ9.hABJPfI6sMVKc09uSyq_3iyl1uWJtLTFmEMua41yAl0)

Polynomial 🡪 O(n), O(n2), O(n3)

Exponential 🡪 O(2n), O(n!)

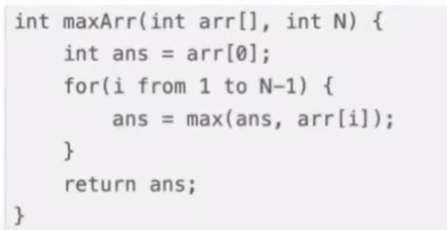
Space Complexity:

Maximum space utilized at any point during running the algorithm.

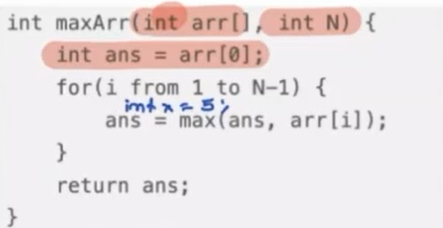


While calculating space complexity we don’t consider input & output space, in above piece of code if we see int N is an input so we ignoring it. Any returning statement also we not consider.

16 bytes is total space in above piece of code. 16 bytes is constant so space complexity is O(1)



Space complexity for above code is O(1)



Space complexity for above code is O (1)

ARRAYS:

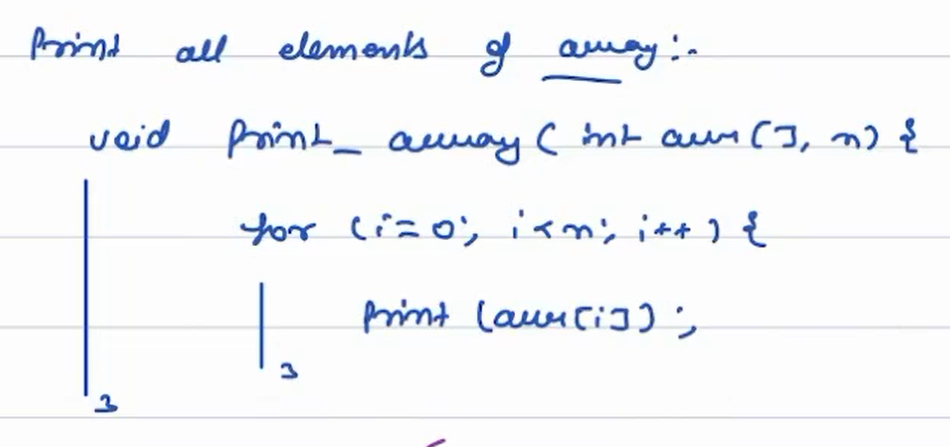
Ordered Collection of same type of data

Array is the collection of similar datatypes which stored at continuous memory location.

Int arr[N];

Data type variable name

arr[0], arr[1], arr[2], …, arr[N-1]



T.C O (n)

S.C O (1)

Q. Given an array of size n, reverse the entire array.

function reverse(int arr[],int n)

{

int i=0, j=n-1;

while (i<j){

int temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

i++;

j--;

}

}

Total number of iterations is n/2 so T.C is O(n) & S.C O(1)

Q Given an array of size n, and given l & r, we have to reverse array from l to r;

function reverse(int arr[],int n, int l,int r)

{

while (l<r){

int temp=arr[l];

arr[l]=arr[r];

arr[r]=temp;

l++;

r--;

}

}

Total number of iterations is n/2 in worst scenario where l=0 and r=n-1 so T.C is O(n) & S.C O(1)

Q. Given an array of size n, rotate it right to left k times.

Brute Force: Easiest and most simple naive solution.

For i=1 to i<=k and i++{

Int temp = arr[N-1]

For j=N-1 to j>=1 and j- -{

Arr[j]=arr[j-1]

}

Arr[0]=temp

}

T.C O(K\*N)

S.C O(1)

Suppose Array of size 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| K | Arr[0] | Arr[1] | Arr[2] | Arr[3] | Arr[4] |
| **Arr Value** | 0 | 1 | 2 | 3 | 4 |
| 1 | 4 | 0 | 1 | 2 | 3 |
| 2 | 3 | 4 | 0 | 1 | 2 |
| 3 | 2 | 3 | 4 | 0 | 1 |
| 4 | 1 | 2 | 3 | 4 | 0 |

Optimize:

K=K%n

Step 1: Reverse whole array (n/2 iterations)

Reverse (arr,N,0,N-1)

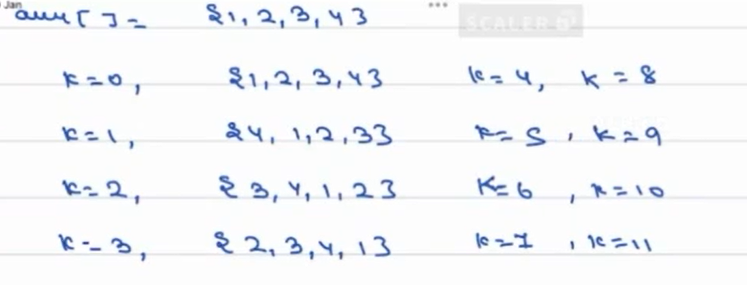
Step 2: reverse 1st k elements (k/2 iterations)

Reverse (arr,N,0,k-1)

Step 3: reverse rest of the elements(range b-a+1 🡪 n-1-k+1 🡪 n-k/2 iterations)

Reverse (arr,N,k,N-1)

Why k=k%n ?



T.C

Total iterations = n/2+k/2+(n-k)/2

= (n+k+n-k)/2

= 2n/2

= n

= O(n)

DYNAMIC ARRAY:

ArrayList is a dynamic array

**For insertion the T.C of Array List is amortized O(1)**

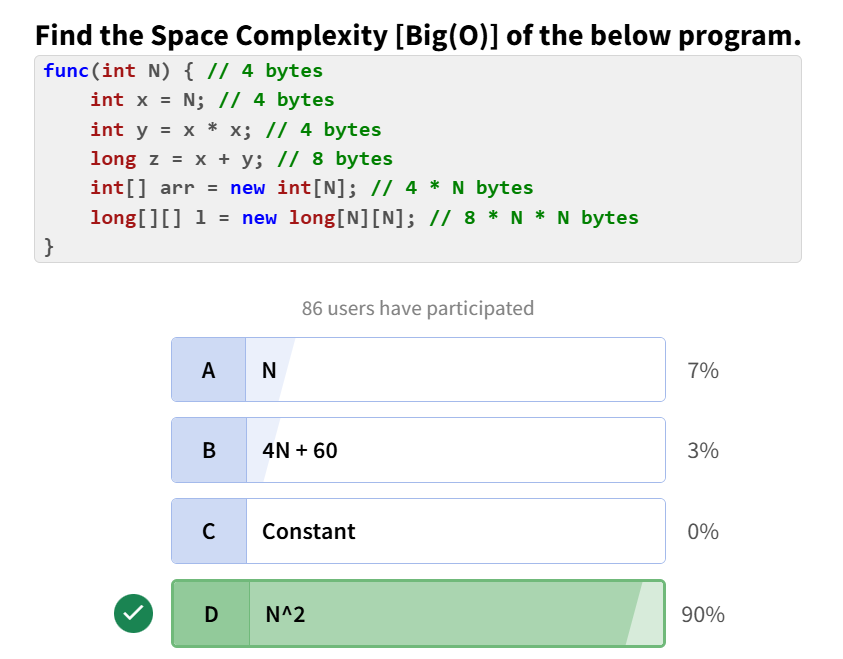
Some operations are cheap some operations are expensive then we called amortized.

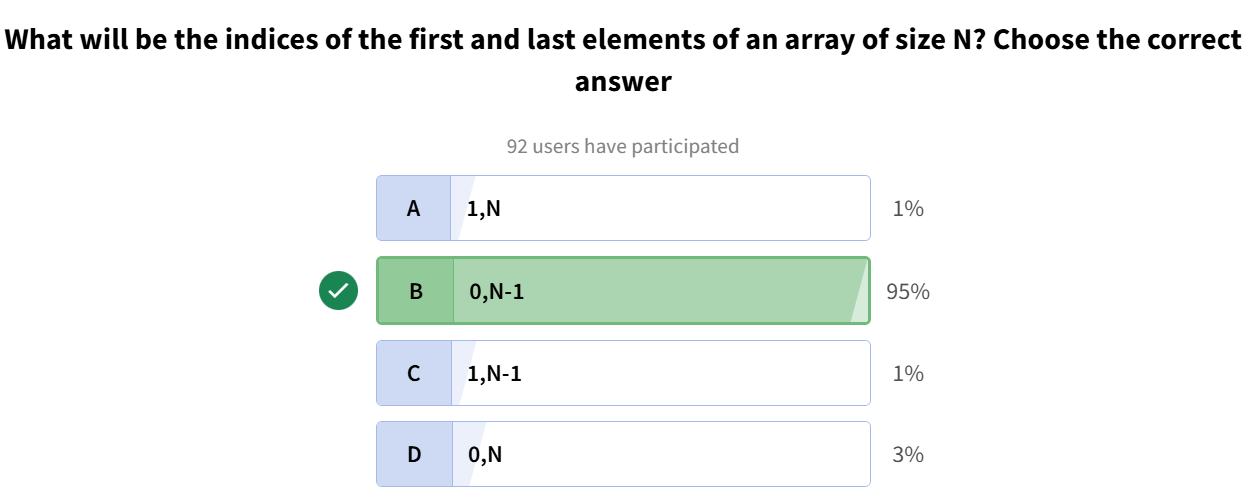
Some operations are expensive and because of this expensive operations other operations are cheap so if we average out all operation we can say its approximately constant O(1) value such scenario called amortized O(1).

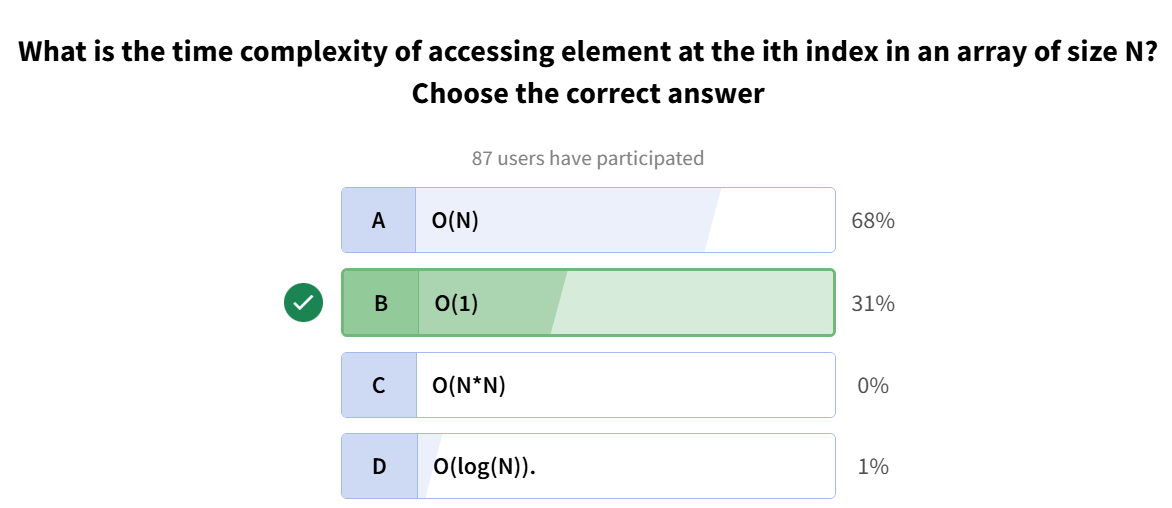
Amortized order of 1 means average of order of 1

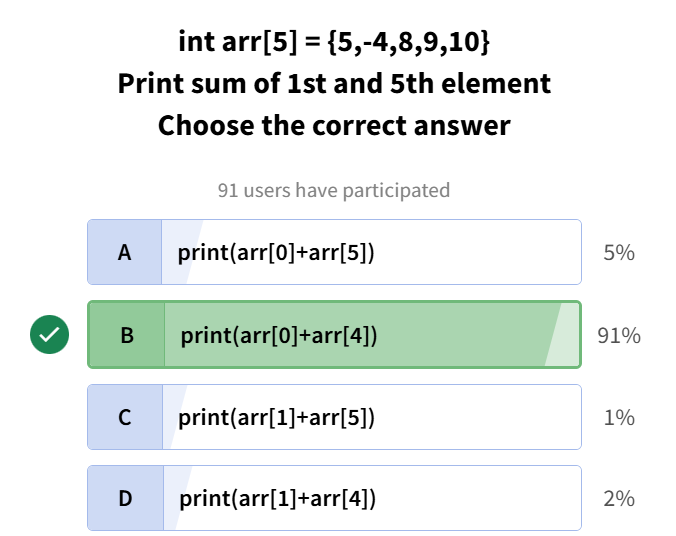
Order of 1 = O (1)

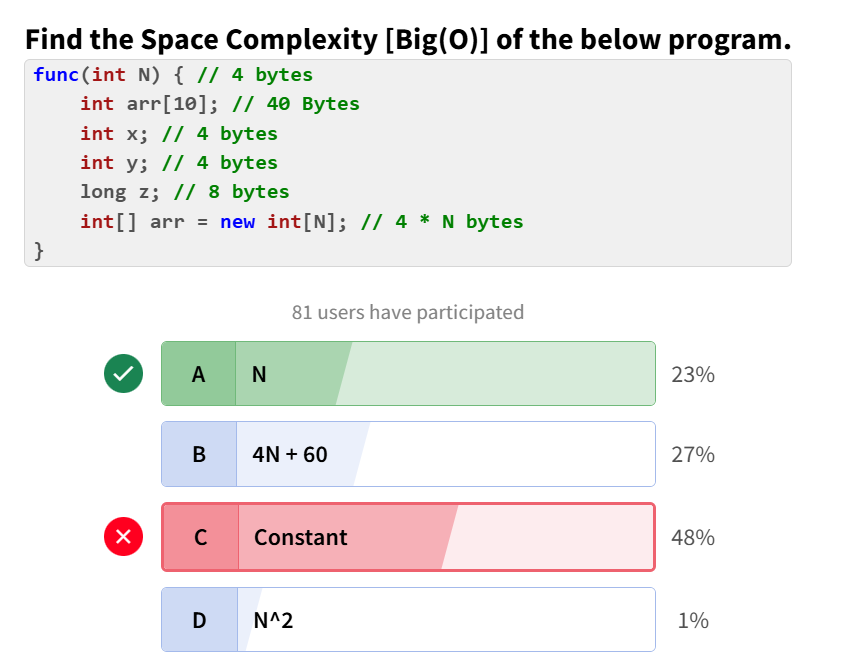
Quizzes









****

**ASSIGNMENTS:**

**Q1. Good Pair**

**Problem Description**

Given an array **A** and an integer **B**. A **pair(i, j)** in the array is a good pair if **i != j** and **(A[i] + A[j] == B)**. Check if any good pair exist or not.

**Problem Constraints**

1 <= A.size() <= 104

1 <= A[i] <= 109

1 <= B <= 109

**Input Format**

First argument is an integer array A.

Second argument is an integer B.

**Output Format**

Return 1 if good pair exist otherwise return 0.

**Example Input**

Input 1:

A = [1,2,3,4]

B = 7

Input 2:

A = [1,2,4]

B = 4

Input 3:

A = [1,2,2]

B = 4

**Example Output**

Output 1:

1

Output 2:

0

Output 3:

1

**Example Explanation**

Explanation 1:

(i,j) = (3,4)

Explanation 2:

No pair has sum equal to 4.

Explanation 3:

(i,j) = (2,3)

**CODE:**

public class Solution {

    public int solve(int[] A, int B) {

        for(int i=0;i<A.length;i++){

            for(int j=0;j<A.length;j++){

                if(i!=j && A[i]+A[j]==B)

                return 1;

            }

        }

        return 0;

    }

}

T.C O(n2)

S.C O(1)

**Q2. Reverse in a range**

**Problem Description**

Given an array **A** of **N** integers and also given two integers **B** and **C**. Reverse the elements of the array **A** within the given inclusive range **[B, C].**

**Problem Constraints**

1 <= N <= 105  
1 <= A[i] <= 109  
0 <= B <= C <= N - 1

**Input Format**

The first argument A is an array of integer.  
The second and third arguments are integers B and C

**Output Format**

Return the array A after reversing in the given range.

**Example Input**

Input 1:

A = [1, 2, 3, 4]

B = 2

C = 3

Input 2:

A = [2, 5, 6]

B = 0

C = 2

**Example Output**

Output 1:

[1, 2, 4, 3]

Output 2:

[6, 5, 2]

**Example Explanation**

Explanation 1:

We reverse the subarray [3, 4].

Explanation 2:

We reverse the entire array [2, 5, 6].

public class Solution {

    public int[] solve(int[] A, int B, int C) {

        while(B<C){

            A[B]=A[B]+A[C];

            A[C]=A[B]-A[C];

            A[B]=A[B]-A[C];

            B++;

            C--;

        }

        return A;

    }

}

T.C O (n)

S.C O (1)

**Q3. Array Rotation**

**Problem Description**

Given an integer array **A** of size **N** and an integer **B**, you have to return the same array after rotating it **B** times towards the right.

**Problem Constraints**

1 <= N <= 105  
1 <= A[i] <=109  
1 <= B <= 109

**Input Format**

The first argument given is the integer array A.  
The second argument given is the integer B.

**Output Format**

Return the array A after rotating it B times to the right

**Example Input**

Input 1:

A = [1, 2, 3, 4]

B = 2

Input 2:

A = [2, 5, 6]

B = 1

**Example Output**

Output 1:

[3, 4, 1, 2]

Output 2:

[6, 2, 5]

public class Solution {

    public static int[] rev(int[] A,int l,int r){

        while(l<r){

            A[l]=A[l]+A[r];

            A[r]=A[l]-A[r];

            A[l]=A[l]-A[r];

            l++;

            r--;

        }

        return A;

    }

    public int[] solve(int[] A, int B) {

        B=B%A.length;

        A=rev(A,0,A.length-1);

        A=rev(A,0,B-1);

        A=rev(A,B,A.length-1);

        return A;

    }

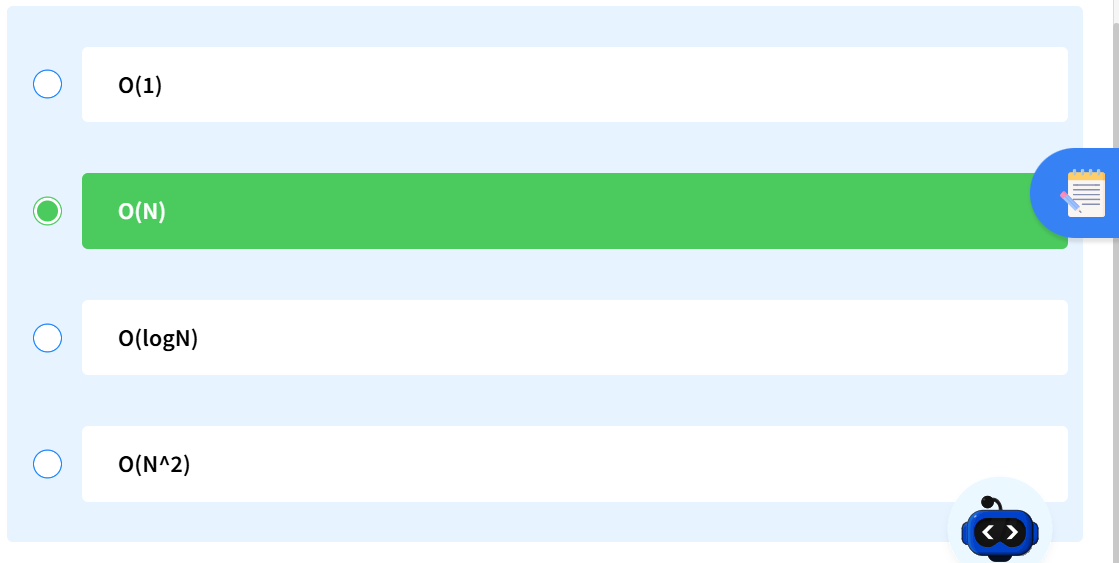
}

T.C O (n)

S.C O (1)

**Q4. Time Complexity – Arrays**

What is the time complexity for inserting/deleting at the beginning of the array?



**Explanation:**

* Arrays are stored in **contiguous memory**.
* To **insert/delete at the beginning**, all existing elements need to be **shifted one position** (right for insert, left for delete).
* Hence, in the **worst case**, this requires shifting **n elements**.

So, **insertion/deletion at the beginning of an array is O(n)**

**Q5. Max Min of an Array**

**Problem Description**

Given an array **A** of size **N**. You need to find the sum of **Maximum and Minimum element** in the given array.

**Problem Constraints**

1 <= N <= 105  
-109 <= A[i] <= 109

**Input Format**

First argument A is an integer array.

**Output Format**

Return the sum of maximum and minimum element of the array

**Example Input**

Input 1:

A = [-2, 1, -4, 5, 3]

Input 2:

A = [1, 3, 4, 1]

**Example Output**

Output 1:

1

Output 2:

5

**Example Explanation**

Explanation 1:

Maximum Element is 5 and Minimum element is -4. (5 + (-4)) = 1.

Explanation 2:

Maximum Element is 4 and Minimum element is 1. (4 + 1) = 5.

public class Solution {

    public int solve(int[] A) {

        int max=Integer.MIN\_VALUE;

        int min=Integer.MAX\_VALUE;

        int i=0;

        int j=A.length-1;

        while(i<=j){

            if(A[i]<min)

            {

                 min=A[i];

            }

            if(A[j]<min)

            {

                min=A[j];

            }

            if(A[i]>max) max=A[i];

            if(A[j]>max) max=A[j];

            i++;

            j--;

        }

        return max+min;

    }

}

Chat GPT:

public class Solution {

public int solve(int[] A) {

int l = 0, r = A.length - 1;

int min = Integer.MAX\_VALUE;

int max = Integer.MIN\_VALUE;

while (l <= r) {

// Handle both l and r in the same iteration

min = Math.min(min, A[l]);

max = Math.max(max, A[l]);

if (l != r) { // Avoid double-counting when l == r

min = Math.min(min, A[r]);

max = Math.max(max, A[r]);

}

l++;

r--;

}

return min + max;

}

}

ADDITIONAL PROBLEMS:

**Q1. Linear Search - Multiple Occurences**

**Problem Description**

Given an array A and an integer B, find the number of occurrences of B in A.

**Problem Constraints**

1 <= B, Ai <= 109  
1 <= length(A) <= 105

**Input Format**

Given an integer array A and an integer B.

**Output Format**

Return an integer, number of occurrences of B in A.

**Example Input**

Input 1:

A = [1, 2, 2], B = 2

Input 2:

A = [1, 2, 1], B = 3

**Example Output**

Output 1:

2

Output 2:

0

**Example Explanation**

Explanation 1:

Element at index 2, 3 is equal to 2 hence count is 2.

Explanation 2:

There is no element equal to 3 in the array.

CODE:

public class Solution {

    public int solve(int[] A, int B) {

        int count=0;

        for(int i=0;i<A.length;i++){

            if(A[i]==B) count++;

        }

        return count;

    }

}

My Second Solution:

public class Solution {

    public int solve(int[] A, int B) {

        int l=0,r=A.length-1;

        int count=0;

        while(l<=r){

            if(l==r && A[l]==B){

                count++;

            }

            else{

            if(A[l]==B)

                count++;

            if(A[r]==B)

                count++;

            }

            l++;

            r--;

        }

        return count;

    }

}

**Q2. Second Largest**

**Problem Description**

You are given an integer array **A**. You have to find the second largest element/value in the array or report that no such element exists.

**Problem Constraints**

1 <= **|A|** <= 105

0 <= **A[i]** <= 109

**Input Format**

The first argument is an integer array **A**.

**Output Format**

Return the second largest element. If no such element exist then return -1.

**Example Input**

Input 1:

A = [2, 1, 2]

Input 2:

A = [2]

**Example Output**

Output 1:

1

Output 2:

-1

**Example Explanation**

Explanation 1:

First largest element = 2

Second largest element = 1

Explanation 2:

There is no second largest element in the array.

public class Solution {

    public int solve(int[] A) {

        int max=Integer.MIN\_VALUE;

        int secondMax=-1;

        for(int i=0;i<A.length;i++){

            if(max<A[i]) max=A[i];

        }

        for(int i=0;i<A.length;i++){

            if(!(A[i]==max) && secondMax<A[i]) secondMax=A[i];

        }

        return secondMax;

    }

}

My Second Code:

public class Solution {

    public int solve(int[] A) {

        if(A.length==1)

            return -1;

        int firstMaxValue=-1;

        int secondMaxValue=Integer.MIN\_VALUE;

        int l=0;

        int r=A.length-1;

        while(l<=r){

            if(firstMaxValue<A[l]){

                secondMaxValue=firstMaxValue;

                firstMaxValue=A[l];

            }else if(secondMaxValue<A[l] && A[l]!=firstMaxValue)

            {

                secondMaxValue=A[l];

            }

            if(firstMaxValue<A[r]){

                secondMaxValue=firstMaxValue;

                firstMaxValue=A[r];

            }else if(secondMaxValue<A[r] && A[r]!=firstMaxValue){

                secondMaxValue=A[r];

            }

            l++;

            r--;

        }

            return secondMaxValue;

    }

}

**Q3. Time to equality**

**Problem Description**

Given an integer array **A** of size **N**. In one second, you can increase the value of one element by 1.  
  
Find the **minimum** time in seconds to make all elements of the array equal.

**Problem Constraints**

1 <= N <= 1000000

1 <= A[i] <= 1000

**Input Format**

First argument is an integer array A.

**Output Format**

Return an integer denoting the minimum time to make all elements equal.

**Example Input**

A = [2, 4, 1, 3, 2]

**Example Output**

8

**Example Explanation**

We can change the array A = [4, 4, 4, 4, 4]. The time required will be 8 seconds.

public class Solution {

    public int solve(int[] A) {

        int max=Integer.MIN\_VALUE;

        int count=0;

        for(int i=0;i<A.length;i++){

            if(max<A[i]) max=A[i];

        }

        for(int i=0;i<A.length;i++){

           count=count+(max-A[i]);

        }

        return count;

    }

}

My second code:  
public class Solution {

    public static int findMax(int[] A,int l,int r){

        int max=Integer.MIN\_VALUE;

        while(l<=r){

            if(max<A[l]){

                max=A[l];

            }

            if(max<A[r]){

                max=A[r];

            }

            l++;

            r--;

        }

        return max;

    }

    public int solve(int[] A) {

        int l=0,r=A.length-1;

        int max=findMax(A,l,r);

        int count=0;

        while(l<=r){

            if(l==r){

                count=count+max-A[l];

            }else{

                count=count+max-A[l];

                count=count+max-A[r];

            }

            l++;

            r--;

        }

        return count;

    }

}

Another optimized way:

public class Solution {

    public int solve(int[] A) {

        int sum = 0;

        int max = 0;

        for(int i=0;i<A.length;i++){

            if(A[i] > max){

                max = A[i];

            }

            sum+=A[i];

        }

        return max\*A.length-sum;

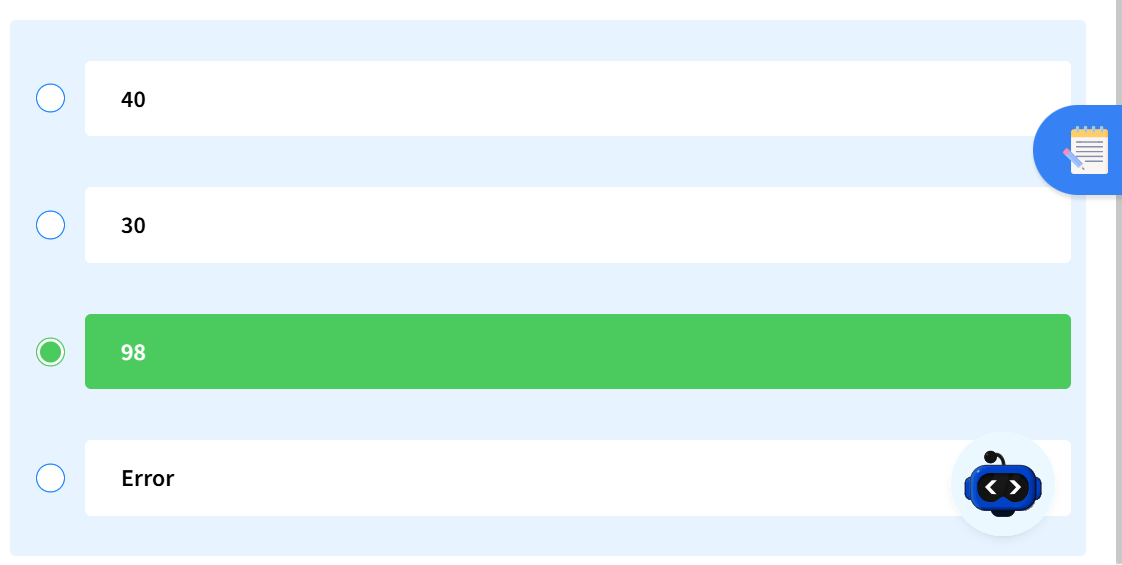
    }

}

**Q4. Problems on 1D Arrays-2 MCQ D**

What will be the output of the following code?

class Main {  
 static void fun(int[]arr) {  
 arr[3] = 98;  
 return;  
 }  
  
 public static void main(String args[]) {  
 int[]arr = {10,20,30,40,50};  
 fun(arr);  
 System.out.println(arr[3]);  
 }  
}



**Q5. Count of elements**

**Problem Description**

Given an array **A** of **N** integers.

Count the number of elements that have at least 1 elements greater than itself.

**Problem Constraints**

1 <= N <= 105  
1 <= A[i] <= 109

**Input Format**

First and only argument is an array of integers A.

**Output Format**

Return the count of elements.

**Example Input**

Input 1:

A = [3, 1, 2]

Input 2:

A = [5, 5, 3]

**Example Output**

Output 1:

2

Output 2:

1

**Example Explanation**

Explanation 1:

The elements that have at least 1 element greater than itself are 1 and 2

Explanation 2:

The elements that have at least 1 element greater than itself is 3

public class Solution {

    public int solve(int[] A) {

        int count=0;

        for(int i=0;i<A.length;i++){

            for(int j=0;j<A.length;j++){

                if(A[i]<A[j]){

                    count++;

                    break;

                }

            }

        }

        return count;

    }

}

My Second Code:

public class Solution {

    public static int findMax(int[] A,int l,int r){

        int max=Integer.MIN\_VALUE;

        while(l<=r){

            if(max<A[l]){

                max=A[l];

            }

            if(max<A[r]){

                max=A[r];

            }

            l++;

            r--;

        }

        return max;

    }

    public int solve(int[] A) {

        int l=0,r=A.length-1;

        int max=findMax(A,l,r);

        int count=0;

        while(l<=r){

            if(l==r){

                if(A[l]<max)

                    count++;

            }else{

                if(A[l]<max)

                    count++;

                if(A[r]<max)

                    count++;

            }

            l++;

            r--;

        }

        return count;

    }

}

Another optimize way:

public class Solution {

    public int solve(int[] A) {

int max = Integer.MIN\_VALUE, count=0;

for(int i=0; i<A.length; i++){

if(max <= A[i]){

if(max == A[i]) count++;

  else{

  max = A[i];

  count = 1;

  }

}

}

return A.length-count;

    }

}